UNIVERSIDADE FEDERAL DO PARANÁ
SETOR DE CIÊNCIAS EXATAS
DEPARTAMENTO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM INFORMÁTICA
ÊNFASE EM TECNOLOGIA DA INFORMAÇÃO

EDIVALDO ANTONIO MONTEIRO

UTILIZAÇÃO DE TÉCNICAS ÁGEIS EM PROJETOS EXCLUSIVOS DE TESTE DE SOFTWARE

CURITIBA
2013

EDIVALDO ANTONIO MONTEIRO

UTILIZAÇÃO DE TÉCNICAS ÁGEIS EM PROJETOS EXCLUSIVOS DE TESTE DE SOFTWARE

Monografia apresentada como requisito parcial à obtenção de grau de Especialista em Informática, ênfase em Tecnologia da Informação, no curso de Pós-Graduação em Informática, Departamento de Informática, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Msc. Nelson Suga

CURITIBA

2013

TERMO DE APROVAÇÃO

Monografia de Especialização
Curso de Especialização em Informática
Ênfase em Tecnologia da Informação
UFPR – ET – Departamento de Informática

Declaramos que o aluno **Edivaldo Antonio Monteiro** entregou a versão final da sua monografia de Especialização em Informática da Universidade Federal do Paraná, com Ênfase em Tecnologia da Informação, intitulada **Utilização de Técnicas Ágeis em Projetos Exclusivos de Testes de Software**.

Curitiba, 25 de maio de 2013.

NELSON SUGA
Professor Adjunto
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 – Curitiba Paraná

SETEMBRINO SOARES FERREIRA JUNIOR
Professor Assistente
Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Informática
Caixa Postal 19081
CEP 81531-990 – Curitiba Paraná

DEDICATÓRIA

Dedico este trabalho à minha família, que foi, é e sempre será o alicerce para todas as edificações da minha vida.

AGRADECIMENTOS

A Deus, por minha vida e por me proporcionar este momento.
A minha esposa, pelo companheirismo e compreensão.
Aos meus pais, por todo o apoio e por acreditar em minha capacidade.
A todos os professores, pela paciência e esforço.
Ao professor Nelson Suga, pelas sugestões apresentadas.
A todos que, direta ou indiretamente, colaboraram para a realização deste trabalho.

Obstáculo é aquilo que você enxerga quando tira os olhos de seu objetivo.

Henry Ford

RESUMO

A área de testes de softwares vem crescendo consideravelmente nos últimos anos. A preocupação com a qualidade dos produtos e serviços ofertados se tornou uma necessidade tão grande que já existem empresas dedicadas única e exclusivamente para testar os sistemas desenvolvidos. Tamanho crescimento e importância requerem técnicas e práticas que facilitem as atividades dos analistas e testadores. É de conhecimento da comunidade científica que as famosas metodologias ágeis possuem técnicas e boas práticas que otimizam os trabalhos de desenvolvimento e gestão de projetos em Tecnologia da Informação. Benefícios como melhoria no relacionamento com os clientes, melhorias na comunicação e integração da equipe e melhoria na qualidade dos produtos e serviços desenvolvidos, são alguns dos ganhos observados em projetos ágeis. No entanto, tais metodologias são comumente aplicadas a projetos de desenvolvimentos de software como um todo. Não há relatos formais conhecidos sobre a aplicação de técnicas ou métodos ágeis em projetos que só atuam na área de testes de software. A finalidade desta pesquisa é avaliar a viabilidade de adoção de técnicas ágeis em projetos exclusivamente dedicados a testes de softwares, demonstrando as melhorias e benefícios que as mesmas podem propiciar a este nicho de negócio. A fonte utilizada como base para as conclusões e resultados obtidos na pesquisa, é oriunda de um estudo de caso (em que o autor foi membro integrante) que foi desenvolvido e aplicado em um projeto de testes de software.

Palavras-chave: Teste de Software. Metodologias Ágeis. Tecnologia da Informação. Métodos Ágeis para Testes.

ABSTRACT

The Software Testing area has been growing substantially over the last years. The concern with quality of products and services offered has increased so significantly that companies dedicated only and exclusively to test the developed systems were created in order to deal with that demand. Such growth and importance requires techniques and practices that make easy the analysts and testers' activities. It is known in the scientific community that the famous Agile Methodologies have techniques and good practices which optimize the activities of development and project management within Information Technology area. Benefits as improvement on Customer relationship, improvement on communication and integrations of team and improvement on quality of products and services developed, are some of gains observed into agile projects. However, these methodologies are commonly applied to software development's Project as a whole. There are no known formal reports about applying agile techniques or methodologies into projects that act only on test software's area. The purpose of this research is to evaluate the feasibility of adopting agile techniques in projects exclusively dedicated to software testing, demonstrating the improvements and benefits that they can provide to this business niche. The source used as basis for the conclusions and results obtained in this research, come from of a case study (in which the author was a member of it) that was developed and applied into a software testing project.

Key-words: Software Test. Agile Methodologies. Information Technology. Agile Methods for Testing.

LISTA DE ILUSTRAÇÔES

LISTA DE SIGLAS

CVS	- *Concurrent Versions System*
FDD	- *Feature Driven Development*
ISTQB	- *International Software Testing Qualifications Board*
RAD	- *Rapid Application Development*
SLA's	- *Service Level Agreement*
TI	- Tecnologia da Informação
VOIP	- *Voice Over Internet Protocol*
XP	- *Extreme Programming*

SUMÁRIO

1 INTRODUÇÃO

Desde os primórdios da engenharia de software, a preocupação com a construção de softwares é ponto relevante, o que explica a existência de diversas metodologias de desenvolvimento de produtos de software, onde padrões, técnicas e métricas são aplicadas. No entanto, as mudanças, comuns em qualquer área, exigiram (e porque não dizer: exigem) a constante melhoria nas formas de desenvolver um produto de software. Neste sentido, destacam-se as chamadas metodologias ágeis, que surgiram com o intuito de tornar a construção de um software mais rápida e menos burocrática em relação às anteriores.

Não distante desta realidade, a preocupação com a qualidade dos produtos de softwares desenvolvidos pelas empresas de tecnologia da informação (TI) está em crescimento constante. Cada dia mais cresce a procura por profissionais qualificados que ajudem a garantir que o produto será entregue o mais próximo possível das necessidades do cliente. Pode-se citar como exemplo deste crescimento o surgimento de empresas que se dedicam única e exclusivamente a testar sistemas desenvolvidos por outras empresas, as chamadas terceirizadas em testes.

No entanto, por maior que seja o crescimento, a área de testes de softwares ainda sofre com as atividades burocráticas e documentação complexa que deve ser gerada para que os objetivos sejam cumpridos. As metodologias ágeis focalizam muito nas fases de desenvolvimento e tratam os testes como parte integrante do processo de construção.

Neste contexto, a aplicação de técnicas utilizadas nas metodologias ágeis em projetos específicos de testes pode contribuir com a melhoria dos processos intrínsecos a estes tipos de projetos? Pretende-se, ao fim deste projeto de pesquisa acadêmica, uma possível resposta a esta pergunta.

1.1 PROBLEMA

A aplicação de metodologias ágeis em testes é fator já conhecido. São perceptíveis na literatura capítulos que falam da aplicação de testes nas fases de desenvolvimento de software. No entanto, as metodologias ágeis são modelos que trabalham, em geral, com projetos pequenos, e focam, em sua maioria, no desenvolvimento das aplicações. Se analisado por esta ótica, subentende-se que, em geral, para grandes projetos, para grandes sistemas, estas metodologias são pouco utilizadas.

Outro fator de relevância que deve ser considerado neste tema é a existência de empresas especializadas em testes que atendem projetos específicos de testes de outras organizações. Para esta área de negócio, pouco se ouve falar a respeito de aplicação de técnicas ágeis como boas práticas para melhorar os processos envolvidos.

Dito isto, esta pesquisa visa analisar o uso de boas práticas utilizadas nas metodologias ágeis em testes de softwares, porém em projetos onde o foco não é o desenvolvimento em si, mas a qualidade do produto final.

1.2 JUSTIFICATIVA

Como profissional da área de testes, não foram poucas as oportunidades de trabalhar em diferentes projetos onde o foco não está na construção e/ou desenvolvimento de um sistema, mas sim nos testes, na qualidade do produto, onde a garantia de um sistema bem testado e funcional é sinônimo de cliente satisfeito e, consequentemente, sucesso no projeto.

Analisando as atuais visões e metodologias referentes a conceitos ágeis adotadas no mercado, é perceptível que não se aplicam diretamente em projetos qualificados neste modelo de negócio. Assim, surge a curiosidade de entender como a adoção de certas “boas práticas” (aplicadas em metodologias ágeis)

podem auxiliar e melhorar os processos internos e/ou terceirizados, utilizados em projetos específicos de testes de software.

1.3 OBJETIVO GERAL

Avaliar de que maneira boas práticas utilizadas em metodologias ágeis de desenvolvimento de softwares podem auxiliar no desempenho de projetos aplicados direta e especificamente em projetos de teste de softwares.

1.4 OBJETIVOS ESPECÍFICOS

- Conhecer, de forma geral, as principais boas práticas utilizadas em metodologias ágeis;
- Conceituar, de forma genérica, testes de software;
- Avaliar possíveis aplicações das técnicas ágeis em projetos específicos de testes de software onde o objetivo principal é a qualidade dos artefatos testados;
- Identificar quais são os possíveis ganhos na adoção de tais técnicas em projetos com o perfil citado;
- Identificar pontos positivos e negativos que técnicas ágeis de desenvolvimento podem trazer à área de testes, se efetuada tal aplicação.

2 REFERENCIAL TEÓRICO

Desenvolver softwares não é uma tarefa fácil. Como em qualquer área da engenharia, metodologias e técnicas existem para padronizar as atividades. A Engenharia de Software, no seu conceito inicial, utilizava metodologias tradicionais, caracterizadas pela forte regulamentação, porém vista por muitos engenheiros de software como um processo burocrático (CARVALHO ET AL., 2011). Esta visão gerou a necessidade de desenvolver métodos ágeis e iterativos, onde o trabalho pudesse ser desenvolvido de forma mais dinâmica e eficiente. Originavam-se aí as chamadas metodologias ágeis.

2.1 METODOLOGIAS ÁGEIS

Metodologias ágeis vêm sendo adotadas como alternativa às abordagens tradicionais de desenvolvimento de software. Segundo Soares (2011), as metodologias tradicionais devem ser aplicadas em situações onde os requisitos são estáveis e os requisitos futuros previsíveis. Entretanto, é cada vez mais comum que os projetos de desenvolvimento de software possuam mudanças, com requisitos que sofrem alterações devido ao dinamismo dos ambientes organizacionais e leis, entre outros fatores. Com equipes pequenas e prazos curtos, há a necessidade de agilidade no desenvolvimento, o que gera a possibilidade de queda na qualidade do produto final. Para tratar estes tipos de ambientes é que surgem as metodologias ágeis.

O termo metodologias ágeis tornou-se popular em meados de 2001, quando especialistas em desenvolvimento de software se reuniram e criaram o “manifesto ágil”, que trata de princípios comuns de diferentes metodologias por eles utilizadas. Foram definidos como conceitos chave do manifesto ágil (AGILE MANIFESTO, 2001):

- Indivíduos e interação entre eles mais que processos e ferramentas
- Software em funcionamento mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos
- Responder a mudanças mais que seguir um plano

Isto quer dizer que, mesmo havendo valor nos itens à direita (processos e ferramentas, documentação abrangente, negociação de contratos, seguir um plano), são valorizados mais os itens à esquerda (indivíduos e interação, software em funcionamento, colaboração com o cliente e responder a mudanças).

Cada metodologia ágil existente apresenta técnicas definidas como boas práticas para um projeto ágil. Algumas delas serão apresentadas as seguir.

2.1.1 *Extreme Programming* (XP)

A XP é uma metodologia ágil em geral utilizada em equipes pequenas e médias e que irão atuar em projetos de softwares pequenos (de 1 a 36 meses), com requisitos vagos e/ou em constantes mudanças (SOARES, 2011). A estratégia adotada é constante acompanhamento e realização de pequenos ajustes durante o desenvolvimento do software. Existe um foco bastante explícito no escopo do projeto. Em outras palavras, são priorizadas funcionalidades que representam maior valor ao negócio. Isto permite que, em casos de necessidade de diminuição de escopo, funcionalidades menos importantes sejam descartadas ou adiadas, sem provocar agravantes no projeto.

O XP apresenta valores baseados em comunicação, *feedback*, coragem, simplicidade e respeito, e princípios básicos definidos em *feedback* rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e alta qualidade do código.

Tal metodologia aplica ainda as seguintes práticas básicas (BARBOSA, 2005):

- Planejamento: Consiste basicamente na definição dos pontos necessários no projeto e o que pode ser adiado. O cliente tem papel fundamental, pois ele ajudará a definir o que é importante para o negócio e o que pode ser colocado em construção em um segundo momento. A cada iteração entregue, novos pontos são definidos para entregas futuras.
- Entregas Frequentes: periodicamente são entregues versões funcionais do projeto. Isso evita surpresas no caso de entrega muito tempo após o estimado e aumenta confiabilidade e aceitação por parte do cliente.
- Metáfora: Basicamente manter a descrição do projeto numa linguagem que seja familiar ao cliente, vislumbrando uma melhoria na comunicação com o mesmo.
- Projeto Simples: O que é feito deve ser simples. Requisitos futuros devem ter seu tratamento efetuado em seu tempo. Em outras palavras, o que deve ser desenvolvido e entregue é exatamente o que o cliente solicitou, qualquer sugestão de melhoria deve ser tratada no futuro em conjunto com outras entregas, ou até mesmo em uma entrega separada.
- Testes: Os testes são criados e executados em paralelo com o desenvolvimento, inclusive com a participação ativa do próprio cliente.
- Programação em Pares: a programação é feita em simultâneo por duas pessoas em uma mesma máquina, enquanto um escreve o código o outro observa, de modo a perceber possíveis erros sintáticos e semânticos e ainda propondo melhoria na execução. Isso possibilita que o programa seja sempre escrito e revisado ao mesmo tempo, além de promover o contínuo aprendizado entre os integrantes da equipe.
- Refatoração: Permite a melhoria contínua da programação. Quando é percebido que determinado módulo pode ser simplificado sem perder nenhuma funcionalidade.

- Posse Coletiva: O código fonte é de todos, assim como a responsabilidade do software com um todo. Isso permite que alterações possam ser feitas por qualquer membro da equipe (desde que devidamente testado), uma vez que todos conhecem todas as partes do sistema.
- Integração Contínua: Sempre que disponível, as funcionalidades devem ser integradas. Isso facilita na detecção de erros de forma prematura, permitindo correção antes da entrega.
- Ritmo Sustentável: Trabalhar com qualidade, buscando um ritmo de trabalho adequado, evitando horas extras (salvo se estas realmente forem agregar produtividade na execução do trabalho).
- Cliente Sempre Presente: O cliente é parte fundamental do processo e deve estar envolvido diretamente nas fases do projeto, seja sanando dúvidas dos requisitos ou processos, seja auxiliando a elaboração de testes. O cliente deve ser visto como parte integrante da equipe de desenvolvimento.
- Código Padrão: padronização da arquitetura do código, de forma que todos integrantes o conheçam e saibam manipulá-lo de forma coesa.

O XP é interessante por uma série de fatores. No entanto, um tópico que chama a atenção é a preocupação com a qualidade do software como um todo, ou seja, em todas as fases existem checagens que são realizadas nos artefatos gerados, com foco em entregar um produto que atenda às necessidades do cliente e mantenha um nível padronizado, que pode ser facilmente identificado.

2.1.2 *SCRUM*

Metodologia ágil utilizada para gestão e planejamento de software. O processo adota ideias da teoria de controle de processos industriais, porém voltadas para o ambiente de desenvolvimento de sistemas. O foco principal é

garantir uma forma de trabalho flexível em ambientes onde a mudança é constante. Ou seja, requisitos, recursos, tecnologia e outras variáveis técnicas podem se tornar obstáculos complexos no ambiente do projeto. Dessa forma, é necessário que o ambiente seja flexível de modo a confrontar os desafios que surgem com o decorrer do tempo. (SOARES, 2011, p.5)

Os princípios do *SCRUM* são semelhantes aos do XP, formado basicamente de equipes pequenas, requisitos instáveis e iterações curtas, para garantir um bom andamento do produto final (que é considerado um produto que o cliente realmente utilize). Um dos diferenciais do *SCRUM* é a forma de organização do escopo e atividades do projeto.

Esta metodologia possui definidos basicamente três papeis, apresentados abaixo:

- *Scrum Master*: pode ser definido como um *mix* de gerente, facilitador e mediador. Assume o papel de remover os obstáculos para a equipe e assegurar que as praticas do *SCRUM* estão sendo aplicadas de forma correta;
- Proprietário do Produto (*Product Owner*): Responsável pela visão de negócio do projeto. Normalmente é um papel desempenhado por indicação do cliente, podendo ser inclusive o próprio cliente;
- Equipe: Grupo multifuncional de pessoas que realizam as tarefas funcionais do projeto (análise, projeto, execução, testes, entre outros).

A metodologia apresenta unidades básicas de desenvolvimento dos requisitos funcionais, chamadas *sprints*. Cada *sprint* tem duração curta (entre uma semana e um mês). É neste período de tempo que as atividades programadas são realizadas e, ao final, as funcionalidades que fazem parte do *sprint* são entregues ao cliente. Antecedem um *sprint* reuniões de planejamento, onde são definidas quais atividades e prazos serão contemplados no *sprint*. Além disso, durante cada s*print* ocorrem reuniões diárias, com duração entre 15 e 30 minutos. O objetivo é manter um canal de comunicação, entre todos os envolvidos no projeto, sobre o progresso das atividades, assim como obstáculos encontrados

(FAGUNDES; DETERS; SANTOS, 2008). Este acompanhamento das atividades auxilia na agilidade de tomadas de decisões e resoluções de problemas.

O *SCRUM* adota práticas que são baseadas nos artefatos que o compõem. Segundo Libardi e Barbosa (2010), estes artefatos estão divididos em:

- *Product Backlog*: Lista de todas as funcionalidades ou mudanças no produto, que são definidas pelo *Product Owner* (ou proprietário do produto). Esta lista é priorizada para refletir a necessidade do cliente e/ou demanda do mercado.
- *Sprint Backlog*: Gerado e definido por toda a equipe, na fase de planejamento do *sprint.* Trata-se de uma lista das funcionalidades ou mudanças do produto, oriundas do *product backlog*, que serão desenvolvidas no próximo *sprint*. Esta lista define também quais serão as atividades a serem realizadas para implementar as funcionalidades definidas para o *sprint*.

Um dos pontos mais importantes (e porque não interessantes) do *SCRUM* são as chamadas *Daily Meeting* (Reuniões diárias). Trata-se de encontros que duram no máximo 15 minutos, onde cada integrante da equipe vai dar um *feedback* referente ao status de suas atividades. De acordo com Libardi e Barbosa (2010), basicamente três tópicos são abordados:

- O que eu fiz ontem?
- O que estou planejando fazer hoje?
- Algo me impede de atingir minha meta?

As reuniões ajudam na integração da equipe com o projeto. Assim, todos podem acompanhar o andamento de cada atividade e saber o que está por vir. Os obstáculos são encontrados com antecedência e têm a atenção voltada a tempo de buscar soluções.

Outras duas técnicas comumente utilizadas no *SCRUM* merecem destaque:

- Reuniões de Revisão *sprint:* Servem para demonstrar o trabalho desempenhado para os interessados do projeto. Antes disso, atividades que foram ou não concluídas são verificadas e/ou revistas. Os participantes normalmente são pessoas ligadas à gestão do projeto e clientes.
- Reuniões de Retrospectiva de *Sprint:* Realizadas com a finalidade de levantar os êxitos e fracassos durante o *sprint*. Toda a equipe participa e os resultados são aplicados para a melhoria contínua do processo e/ou do projeto.
- Lembretes: Trata-se da utilização de *post-its* com anotações referentes às atividades relativas a um determinado período. Estas anotações servem para auxiliar os profissionais quanto à organização e andamento de suas atividades. Os lembretes são criados pelo próprio responsável da atividade e fixados em quadros de acompanhamento. Abordagens mais recentes de metodologias ágeis utilizam o *Lean Kanban*[1] como ferramenta para este fim.

O *SCRUM* é apontado como a metodologia ágil mais popular na atualidade. Paulk (2013) realizou recente pesquisa onde mostra a atual abordagem do *SCRUM* nas empresas. Como pode ser observado na FIGURA 1, das empresas pesquisadas e que adotam o *SCRUM*, telecomunicações e serviços financeiros são os setores que estão mais agressivos nesta abordagem, enquanto que companhias públicas e aeroespaciais ainda relutam na adoção do *SCRUM*.

[1] Ferramenta visual, de baixo custo, utilizado para evitar faltas de materiais e impossibilitar a superprodução (produzir mais que o necessário, antes ou depois da hora) (LIB, 2010). Vem sendo adaptada para utilização em projetos ágeis com o intuito de organizar as atividades de projetos de software.

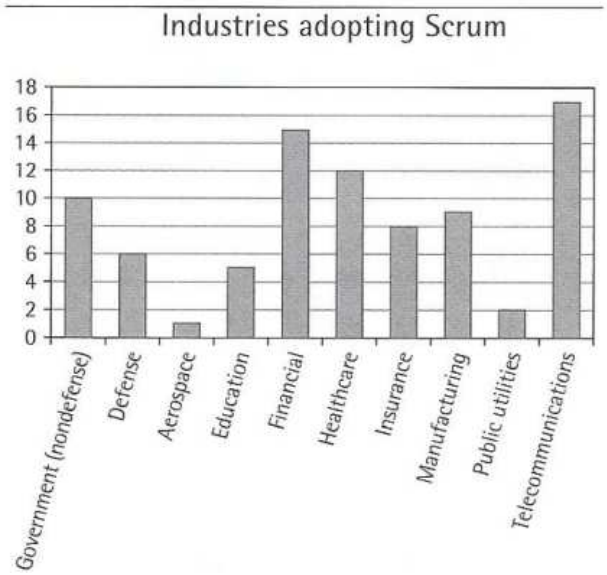

FIGURA 1: SETORES DA INDÚSTRIA QUE UTILIZAM O *SCRUM*.
FONTE: Paulk (2013, p. 31).

Como citado no início desta seção, o *SCRUM* é comumente indicado para equipes pequenas. Porém esta visão está sendo ultrapassada e pode-se verificar (conforme FIGURA 2 abaixo) que equipes maiores estão trabalhando com a metodologia. Equipes com 16 ou mais integrantes já representam um percentual considerável entre todas que utilizam o *SCRUM* como metodologia ágil.

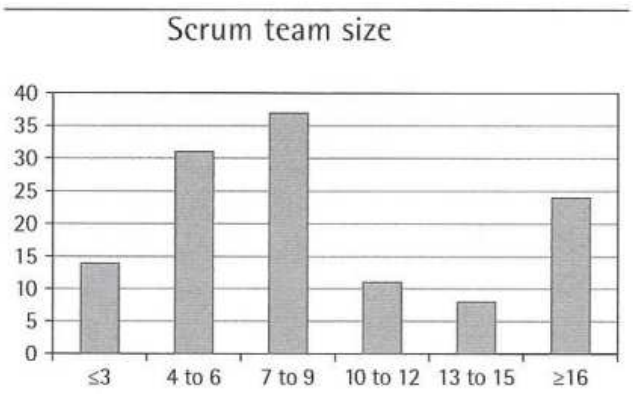

FIGURA 2: TAMANHO DE EQUIPES *SCRUM*.
FONTE: Paulk (2013, p. 31).

A pesquisa de Paulk (2013) traz ainda um ponto bastante vantajoso que o *SCRUM* está propiciando a seus adeptos. Quando comparado com outras metodologias, o *SCRUM* apresentou melhor qualidade nos produtos

desenvolvidos e obteve melhor média de satisfação do cliente final, sem contar o fato de tender a ser menos custoso que seus concorrentes (FIGURA 3).

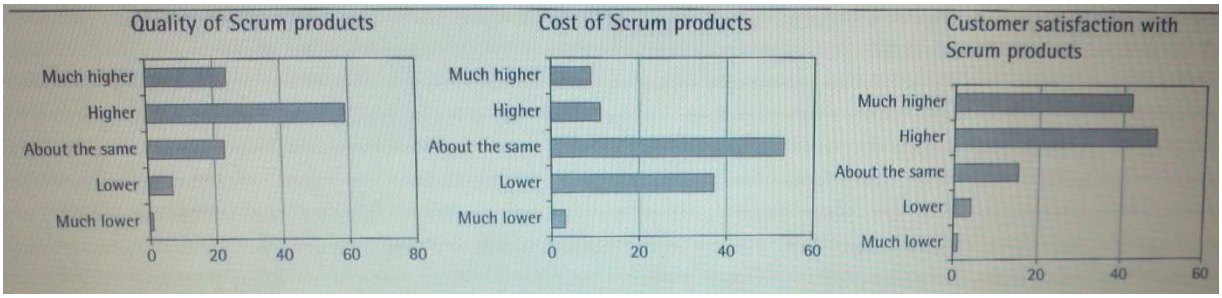

FIGURA 3: QUALIDADE, CUSTO E SATISFAÇÃO DOS CLIENTES COM PRODUTOS *SCRUM*.
FONTE: Paulk (2013, p. 33).

2.1.3 FDD (*Feature Driven Development*)

Segundo Camara (2007), a FDD foi nascida a partir da experiência de análise e modelagem orientadas por objetos de Peter Coad, e de gerenciamento de projetos de Jeff De Luca. Trata-se de uma das seis metodologias ágeis originais (os representantes dela ajudaram a redigir o manifesto ágil). De acordo com Heptagon (2012), o lema principal de dita metodologia é “Resultados frequentes, tangíveis e funcionais”. Ela permite desenvolver sistemas de forma rápida, podendo facilmente ser adicionadas funcionalidades no decorrer (ou até mesmo depois de concluído) o desenvolvimento.

A FDD chama a atenção por algumas características peculiares, que são apresentadas pela Heptagon (2012) como:

- Resultados úteis a cada duas semanas ou menos;
- Blocos pequenos de funcionalidade valorizada pelo cliente, chamados "Features”;
- Planejamento detalhado e guia para medição;
- Rastreabilidade e relatórios com precisão;
- Monitoramento detalhado dentro do projeto, com resumos de alto nível para clientes e gerentes;

- Fornece uma forma de saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa são sólidos.

Segundo Barbosa *et al.* (2011), a FDD constitui-se basicamente de cinco processos, que definem a maneira de trabalho adotada pela metodologia. Abaixo segue uma descrição resumida de cada um.

2.1.3.1 Desenvolver um Modelo Abrangente

É a atividade inicial que irá abranger o projeto como um todo. Todos os envolvidos no projeto participam desta fase, pois é nela que serão definidos os escopos. Todo o estudo é realizado com base no domínio do negócio, para cada área a ser modelada. Após o estudo são formados pequenos grupos, normalmente formados por desenvolvedores, onde cada grupo apresenta um modelo para satisfazer o escopo. São selecionados os melhores modelos e realizada a junção. Ao final desta fase um modelo de objetos é criado, e será base de consulta para a equipe durante a construção do produto.

2.1.3.2 Construir uma Lista de Funcionalidades

Atividade onde é realizada a decomposição do modelo de negócio. A divisão normalmente é realizada por áreas, atividades de negócio e passos da atividade. O resultado final é uma lista de funcionalidades que irão compor o produto final, dispostas hierarquicamente por importância.

2.1.3.3 Planejar por Funcionalidade

Fase onde o planejamento é realizado com base nas funcionalidades, levando em consideração a complexidade e dependência entre elas. O resultado é o plano de desenvolvimento com os prazos e atribuição das atividades definidos. Em geral, as atividades são atribuídas para um programador líder, que fará a distribuição das mesmas na fase seguinte.

2.1.3.4 Detalhar por Funcionalidade

Esta fase já pode ser considerada como construção do produto. A equipe detalha, para cada funcionalidade, os requisitos e outros artefatos para a codificação. Os testes também já começam a ser definidos nesta etapa. É efetuada uma inspeção nas funcionalidades do projeto, principalmente no design. O resultado é um pacote de design do produto, com o esqueleto do código fonte criado, inclusive.

2.1.3.5 Construir por Funcionalidade

Os códigos são preenchidos com as validações finais, testados e inspecionados. Ao término das atividades, o código é integrado às demais funcionalidades para a criação do produto final, com potencial para entregas ao cliente.

O FDD utiliza também várias práticas que podem ser observadas em outros métodos ágeis como testes unitários, refatoração, programação em pares, integração contínua e outras.

2.1.4 RAD (*Rapid Application Development*)

Significa Desenvolvimento Rápido de Aplicação. É um modelo de processo de desenvolvimento de software incremental utilizado para desenvolvimento de aplicações em um tempo bastante curto, normalmente entre 60 e 90 dias (PINHEIRO *et al.*, 2007).

O modelo, apesar de ser linear, enfatiza agilidade no desenvolvimento de aplicações. Isso se torna possível devido à abordagem de construção baseada em componentes. O modelo é usualmente utilizado quando existem requisitos bem definidos e escopo restrito.

O RAD é divido nas seguintes fases:

- Modelagem de Negócio: Levantamento dos processos e fluxo de informação que irão compor a aplicação.
- Modelagem dos Dados: Definição dos objetos de dados a serem processados pela aplicação, assim como as relações, de acordo com o fluxo de informação definido.
- Modelagem do Processo: Os objetos definidos na fase anterior são transformados. Descrições dos processos são criadas por meio da manipulação dos objetos de dados.
- Geração da Aplicação: A aplicação é gerada utilizando ferramentas automatizadas como Delphi, Visual Basic, etc., para facilitar a construção da aplicação.
- Teste e Modificação: São efetuados testes e correção de defeitos, mesmo nos componentes que já passaram por baterias de testes (devido ao reuso de componentes).

Segundo Maner (1997), o RAD funciona bem principalmente em projetos rápidos, mas tende a falhar em alguns casos, por exemplo quando a aplicação necessita de um alto grau de integração com outros sistemas ou quando a distribuição do produto será em grande escala. Nestes casos, a utilização de tal modelo não é aconselhada.

2.1.5 Estado de Uso das metodologias ágeis

Cada vez mais a comunidade de desenvolvimento de software vem adotando metodologias ágeis como forma de melhorar processos e resultados. É perceptível o aumento da quantidade de ferramentas ágeis e sua utilização por empresas de tecnologia da informação.

Uma prova disso está bem especificada no resultado da sétima pesquisa anual de estado do desenvolvimento ágil, elaborado e divulgado pela Versionone (2013). A pesquisa foi respondida por 4.048 pessoas de diferentes canais dentro da comunidade de desenvolvimento de software. De acordo com a pesquisa, o momento é favorável às metodologias ágeis. Dentre os pesquisados, 83% responderam que utilizam ou planejam utilizar tais metodologias no futuro.

Dentre os dados divulgados na pesquisa, alguns chamam a atenção e podem bem exemplificar o crescimento e perfil de adeptos aos métodos ágeis, como:

- Do total de entrevistados, 81% conhecem ou já tiveram ao menos um contato com metodologias ágeis;
- O *SCRUM* (e suas variáveis) é utilizado por 72% das empresas, sendo, dessa forma, a metodologia mais popular;
- A utilização do *Kanban* (e suas variáveis) duplicou em relação ao ano anterior, isso devido também ao crescimento da adoção do *SCRUM*;
- As técnicas ágeis mais utilizadas são as reuniões diárias, planejamento por iterações e testes unitários. No entanto, as técnicas que tiveram maior crescimento em relação ao ano anterior foram o *Kanban* e a retrospectiva;
- As maiores preocupações em adotar técnicas ágeis são: a falta de planejamento adiantado, perda do controle de gestão, oposição da gestão e falta de documentação;
- Como maiores benefícios da utilização de metodologias ágeis, destacaram-se: habilidade de gerir mudanças prioritárias, ganho de

produtividade, aumento na visibilidade do projeto, melhora na moral da equipe e aumento da qualidade dos softwares desenvolvidos.

2.2 TESTES DE SOFTWARE

A produção de software tem crescido significativamente nas últimas décadas. Cada vez mais se tem procurado maneiras de otimizar tarefas, com o intuito de obter ganhos referentes a tempo, produtividade, qualidade dentre outros fatores. A tecnologia da informação, de maneira geral, contribui para este feito. O grande objetivo é facilitar o dia a dia, utilizando softwares como ferramentas de apoio para a otimização de atividades diárias, que demandam tempo e custos elevados.

Desde os primeiros modelos de metodologias de desenvolvimento (modelo em cascata, por exemplo), as atividades de testes já eram incluídas como parte integrante do processo de construção de um software. Isto porque a aplicação de testes de software está diretamente ligada ao nível de qualidade do produto final. Um software com pouco ou nenhum teste tem grandes chances de ser recusado pelo usuário final por problemas de qualidade, colocando em risco todo o projeto a ele vinculado.

O custo relativo à fase de testes dentro de um projeto também é fator preponderante para a sua existência. Pressman (2009) afirma que o custo da fase de testes dentro de um projeto pode variar de acordo com o tipo de software e o processo que está sendo desenvolvido. Por exemplo, para sistemas tradicionais desenvolvidos em modelo cascata, o custo de uma fase de testes dentro do ciclo de desenvolvimento gira entre 40 e 50% do custo total.

Na FIGURA 4 é possível visualizar a variação de custos de testes em diferentes modelos de desenvolvimento de software.

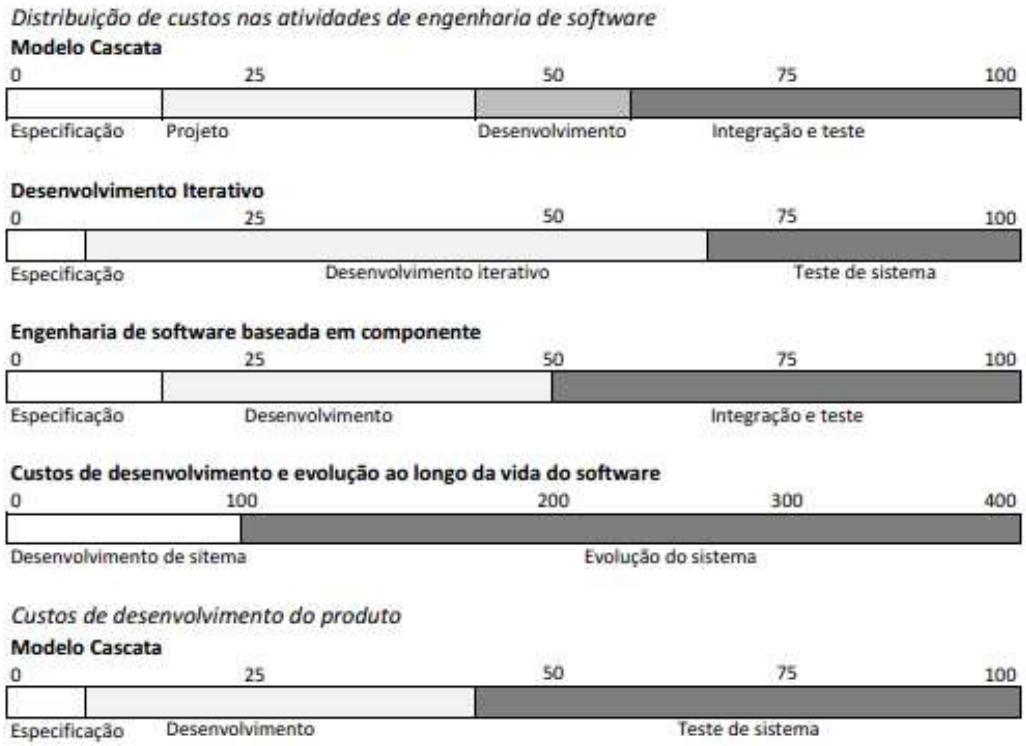

FIGURA 4: DISTRIBUIÇÃO DE CUSTO NAS ATIVIDADES DE ENGENHARIA DE SOFTWARE.
FONTE: Pressman (2009, p. 8)

Sommerville (2007, p.9) afirma que, para sistemas tradicionais, o custo referente a testes representa ao redor de um quarto ou mais do custo de todo o ciclo. Ele aborda ainda a revisão técnica formal como ferramenta de descoberta precoce de defeitos e apresenta um quadro comparativo (representado na FIGURA 5). Nele é fácil perceber que o custo final do projeto sem a utilização da revisão formal é quase três vezes maior quando comparado com o mesmo projeto adotando a revisão.

Erros encontrados	*Número*	*Custo Unitário*	*Total*
	Revisões Realizadas		
Durante o projeto	22	1,5	33
Antes do teste	36	6,5	234
Durante o teste	15	21	315
Após o lançamento	3	67	201
			783
	Nenhuma revisão realizada		
Antes do teste	22	6,5	143
Durante o teste	82	15	1,230
Após o lançamento	12	67	804
			2,177

FIGURA 5: COMPARAÇÃO DOS CUSTOS DE DESENVOLVIMENTO.
FONTE: Sommerville (2007, p. 740)

Mas afinal, o que é teste de software? Existem várias definições que descrevem o conceito de teste de software, no entanto todas acabam, de modo geral, direcionando a uma palavra chave: “qualidade”.

Teste de software, segundo Neto (2007), é o processo de execução de um produto de modo a garantir que as funcionalidades estão em correto funcionamento, baseado nas especificações do produto. O objetivo é identificar falhas ou defeitos no produto antes da entrega ao usuário final.

Porém existem outras definições onde englobam o teste não somente como uma fase de execução. Segundo o *International Software Testing Qualifications Board* – ISTQB (2011), a fase de execução de testes é somente uma parte de um todo maior referente a atividades de testes. Existem atividades antes e depois da execução de testes. Planejamento e controle, escolha das condições de testes e modelagem são exemplos de fases anteriores, assim como checagem de resultados, avaliação de critérios de conclusão e geração de relatórios de conclusão são atividades posteriores à execução de testes.

O ISTQB (2011) também define diferentes objetivos relativos à fase de testes, como:

- Encontrar defeitos.
- Ganhar confiança sobre o nível de qualidade.
- Prover informações para tomada de decisões.

- Prevenir defeitos.

Diferentes pontos de vista levam a diferentes objetivos. Por exemplo, no teste feito em desenvolvimento (teste de componente, integração e de sistemas), o principal objetivo pode ser causar o maior número de falhas possíveis, de modo que os defeitos no software possam ser identificados e resolvidos. No teste de aceite o objetivo principal pode ser confirmar se o sistema está funcionando conforme o esperado, ou seja, prover a confiabilidade de que esteja de acordo com o requisito. Em alguns casos o principal objetivo do teste pode ser avaliar a qualidade do software (não com a intenção de encontrar defeitos), para prover aos gestores informações sobre os riscos da implantação do sistema em um determinado momento. Os testes de manutenção podem ser usados para verificar se não foram inseridos erros durante o desenvolvimento de mudanças. Durante os testes operacionais, o principal objetivo pode ser avaliar características como confiabilidade e disponibilidade.

2.2.1 Técnicas de Testes de Software

Assim como os objetivos de testes, as técnicas podem variar de acordo com cada projeto. Existem inúmeras maneiras de testar um software, porém algumas técnicas são utilizadas com mais frequência (mesmo com as diferenças de paradigmas de desenvolvimento existentes).

> Classificar testes como caixa-preta ou caixa-branca é uma diferenciação clássica. Técnicas caixa-preta (também chamadas de técnicas baseadas em especificação) são uma forma de derivar e selecionar as condições e casos de testes baseados na análise da documentação. Isto inclui testes funcionais e não funcionais para um componente ou sistema, sem levar em consideração a sua estrutura interna. Técnicas de caixa branca (também chamadas de técnicas estruturais ou baseadas em estrutura) são baseadas na estrutura interna de um componente ou sistema (ISTQB, 2011).

Para o objetivo do presente documento serão adotadas como base de estudos as técnicas de caixa-branca, caixa-preta, caixa-cinza e técnicas não funcionais. Ditas técnicas serão apresentadas a seguir.

2.2.1.1 Técnica Estrutural (Caixa-Branca)

Segundo ISTQB (2011), os casos de testes são derivados de informações de como o software é construído. São utilizados componentes gerados na fase de construção do produto (exemplo: código fonte e modelagem). Em outras palavras, a técnica avalia o comportamento interno dos componentes do software. Aspectos como teste de condição, teste de fluxo de dados, teste de ciclos, teste de caminhos lógicos, códigos nunca executados são abordagens avaliadas dentro da técnica de caixa branca.

Normalmente quem executa os testes nesta técnica é o próprio desenvolvedor, ou integrante da equipe de desenvolvimento. Isto porque o programador já está familiarizado com o código fonte e pode aplicar os testes com maior agilidade.

2.2.1.2 Técnica Funcional (Caixa-Preta)

A técnica funcional (também conhecida como baseada em especificação) é aplicada de modo a avaliar o comportamento externo do software, não levando em consideração nenhuma abordagem a aspectos internos do mesmo. Os modelos, sejam eles formais ou não, podem ser utilizados como base de execução dos testes (ISTQB, 2011).

Em outras palavras, a técnica de caixa-branca prioriza em testar os requisitos funcionais do software utilizando dados de entrada com saídas esperadas. Uma funcionalidade pode ser testada sabendo-se que para determinado conjunto de entrada de dados, uma saída específica deverá ser

gerada. O objetivo principal é certificar-se de que a funcionalidade realmente faz o que se espera. Isto pode acarretar em certo risco para o sistema, uma vez que, se a documentação de requisitos funcionais não estiver bem detalhada, pode levar a ambiguidade no entendimento das funcionalidades. No entanto é extremamente importante, uma vez que este teste garante um nível mínimo de qualidade em termos funcionais.

2.2.1.3 Técnica Caixa-Cinza

É uma mescla das técnicas de caixa-branca e caixa-preta. Trata-se de avaliar o comportamento funcional do software, porém considerando o correto funcionamento dos componentes internos do sistema. É uma abordagem pouco conhecida, no entanto completa, uma vez que cobre algumas deficiências que as técnicas que a compõem não apresentam quando adotadas de maneira isolada (ISTQB, 2011).

2.2.1.4 Técnicas não funcionais

Existem técnicas de testes que não estão diretamente relacionadas às funcionalidades, mas que são importantes para o projeto como um todo. Trata-se de avaliar aspectos que, mesmo não ligados diretamente ao propósito do software, irão influenciar diretamente a aceitação do mesmo pelo cliente. Para esta técnica podemos citar alguns níveis de testes, como (ISTQB, 2011):

- Teste de desempenho e carga: avalia a capacidade de processamento do software sob grande demanda de dados e dentro de padrões de tempos pré-definidos como aceitáveis;
- Usabilidade: avalia quão fácil é para os usuários finais o aprendizado e utilização do software. Envolve compreensibilidade

das mensagens, integridade visual das interfaces, entre outros aspectos;

- Confiabilidade: utilizado para avaliação de aspectos ligados à segurança, garantindo sigilo das informações armazenadas/processadas.
- Recuperação: utilizado para verificar a confiabilidade e robustez do sistema em retornar a um estado estável de execução após apresentar um estado de falha.

2.2.2 Tipos de testes

Alguns tipos de testes podem ser aplicados claramente em uma única técnica, outros têm elementos de mais de uma técnica (testes integrados, por exemplo, podem ser classificados tanto como caixa-branca como caixa-preta). Alguns autores qualificam os tipos de testes como níveis de testes, uma vez que, dependendo do planejamento, deve ocorrer em diferentes níveis e em paralelo ao desenvolvimento do software (ROCHA; MALDONADO; WEBER, 2001).

Cada tipo de teste tem foco em um objetivo particular, que pode ser teste de uma funcionalidade a ser realizada pelo software, uma característica da qualidade não funcional, tal como a confiabilidade ou usabilidade, a estrutura ou arquitetura do software ou sistema ou mudanças relacionadas (ISTQB, 2011). Em seguida serão apresentados alguns tipos de testes.

2.2.2.1 Teste de Unidade

Também conhecido como teste unitário ou de módulo. Neste tipo de teste são avaliadas pequenas partes do software. O objetivo principal é encontrar falhas em componentes, objetos, pequenos trechos do código, entre outros. Pode ser

utilizado tanto como técnica de caixa-branca como de caixa-preta. São testes rápidos, normalmente executados pelo próprio desenvolvedor.

2.2.2.2 Teste de Integração

O teste de integração é caracterizado por testar as interfaces entre os componentes, interações de diferentes partes de um sistema, como o sistema operacional, arquivos, hardware ou interfaces entre os sistemas. Conforme ISTQB (2011), este tipo de teste pode ser aplicado em várias fases, por exemplo:

- Teste de integração de componente: testa interações entre componentes de software e é realizado após o teste de componente.
- Teste de integração de sistemas: testa interação entre diferentes sistemas e pode ser realizado após o teste de sistema.

De forma geral, o objetivo principal é garantir que tanto dados quanto processos de integração estejam livres de falhas. Por exemplo, a integração de dados resultantes de um Módulo A que são dados de entradas para um Módulo B, ou ainda dados gerados por um sistema X, que servirão de entrada para um sistema legado Y.

2.2.2.3 Teste de Sistema

O teste de sistema faz alusão ao sistema como um todo, ou seja, é o produto de software o qual o projeto referencia. Neste tipo de teste todas as variáveis devem seguir o mais próximo a um ambiente de produção. Busca-se encontrar falhas nas funcionalidades do sistema utilizando ambiente, massa de dados, interfaces e todas as outras variáveis possíveis, como se o próprio usuário final estivesse utilizando o software. O objetivo é evitar que falhas funcionais

passem despercebidas. Teste de sistema deve tratar requisitos funcionais e não funcionais do sistema. Dessa forma pode ser utilizado tanto em caixa-branca como em caixa preta.

2.2.2.4 Teste de Aceite

Frequentemente realizado pelo cliente ou usuário do sistema. O objetivo principal não é encontrar defeitos, mas sim estabelecer um nível de confiança para o cliente antes da entrega definitiva do sistema. Irá garantir que o sistema está apropriado para uso. No teste de aceite existe uma diferenciação definida pelo ISTQB (2011) como Alfa e Beta de testes, onde no Alfa os testes ocorrem no local onde o produto foi desenvolvido e no Beta (ou teste no campo) é realizado nas dependências do cliente (usuário).

2.2.2.5 Teste de Regressão

É considerado para alguns autores não como um tipo (ou nível) de teste, mas como estratégia de redução de riscos para o sistema. Trata-se de, a cada nova release, ciclo ou versão do software, aplicar todos os testes que já foram aplicados anteriormente. É uma forma de garantir que alterações no código ou em funcionalidades não afetaram funcionalidades que permaneceram intactas. Normalmente é realizado por *scripts* automatizados.

2.2.2.6 Teste Exploratório

A definição de testes exploratórios, segundo Bach (2003), é aprender, desenhar e executar testes de maneira simultânea ou concorrente. Na visão de

Tinkham e Kaner (2003), testes exploratórios envolvem ao mesmo tempo aprendizado, planejamento, execução de testes e abertura de falhas. Ou, como define Caetano (2011): é a criação e execução, ao mesmo tempo, de um caso de teste.

Em outras palavras, teste exploratório é uma abordagem de teste em que o testador não segue *scripts* ou passos específicos definidos anteriormente, mas busca, por meio da experiência e “curiosidade”, explorar as funcionalidades de um sistema, no intuito de encontrar a maior quantidade de falhas possíveis. Os casos de testes, assim como cenários de testes, são praticamente definidos e criados em tempo real, de acordo com conhecimento e aprendizado. Neste tipo de teste, o testador tem a liberdade de ser criativo e utilizar variadas possibilidades na busca por uma falha.

Os testes exploratórios buscam investigar e encontrar as fraquezas de um software, e são utilizados em situações onde:

- Há pouca ou nenhuma especificação ou requisitos do sistema;
- Não há tempo suficiente para especificar, criar e executar testes;
- Possui-se recursos com elevado grau de conhecimento do software a ser testado;
- Necessita-se isolar e investigar um defeito particular;
- Validar a usabilidade de um sistema;
- Em projetos ágeis;
- Em sistemas desconhecidos, instáveis ou que nunca foram testados.

3 METODOLOGIA

A pesquisa é uma forma de encontrar respostas para questões propostas, utilizando para isso métodos científicos. Toda pesquisa implica em um levantamento de dados provenientes de várias fontes, seja qual for o método ou técnica empregada (MARCONI; LAKATOS, 2001). O tipo de pesquisa utilizado para o levantamento de dados deste documento foi a pesquisa bibliográfica ou pesquisa de fontes secundárias, definida por Marconi e Lakatos (2001) como consultas a bibliografias já publicadas em forma de livros, revistas, publicações avulsas e imprensa escrita. A principal finalidade da pesquisa bibliográfica é “colocar o pesquisador em contato direto com tudo aquilo que foi escrito sobre determinado assunto” (MARCONI; LAKATOS, 2001).

A presente pesquisa traz um estudo de caso onde foi possível realizar uma observação da aplicação de boas práticas ágeis em projeto específico de teste de software. Para definição de resultados foi utilizada tanto a experiência própria do autor quanto de outras pessoas envolvidas no projeto de testes que serviu de estudo de caso para esta pesquisa. Em dito estudo, de maneira a ter uma visão mais abrangente, foram coletadas opiniões de pessoas que participaram diretamente do projeto e ajudaram a sinalizar os pontos fortes e fracos que compõem o resultado final do estudo.

A técnica prática de coleta de dados aplicada foi a documentação direta. Esta se subdivide em observação direta intensa, onde foi utilizada a técnica da entrevista – conversação face a face, proporcionando ao entrevistador verbalmente a informação necessária (MARCONI; LAKATOS, 2001).

Através de pesquisas efetuadas por meio eletrônico (internet), foi possível a coleta de materiais acadêmicos e não acadêmicos, onde foram encontradas características que auxiliaram na montagem do conteúdo apresentado, sendo possível atingir o resultado final abordado neste projeto.

4 ESTUDO DE CASO[2]

A WSA (nome fictício) é uma multinacional da área de tecnologia da informação que atua com consultoria e implantação de sistemas para varejo. Os projetos de implantação executados pela WSA, em sua grande maioria, possuem um determinado grau de customização, a fim de que o software possa se adequar corretamente aos processos do cliente.

Os projetos de customização, por se tratar de funcionalidades novas, recebem um foco especial relativo à qualidade. Assim, a fase de testes é considerada como um subprojeto, ou seja, é tratada como se fosse um projeto à parte, com pessoas dedicadas única e exclusivamente para as atividades relacionadas a testes.

A área de testes de software da WSA é composta por profissionais com diferentes níveis de experiência, porém em contínuo aprendizado com o software, não apenas no aspecto técnico, mas também no aspecto processual (regras de negócio). A empresa conta ainda com metodologia de análise e desenvolvimento de sistemas definida, seguindo rigorosos processos de auditoria. A metodologia utilizada para testes é tradicional, e em determinadas atividades a burocracia acaba ”dificultando” o bom andamento da atividade, em alguns casos gerando pequenos atrasos.

Uma vez que os processos formais já são definidos e utilizados pela matriz há muito tempo, qualquer alteração ou inclusão de processo seria extremamente complicada e burocrática. Surgiu então a ideia de, informalmente, otimizar algumas das tarefas com o objetivo de alcançar ganhos de desempenho e produtividade em testes.

Em um projeto cuja fase de testes estava a caminho de ser iniciada, foi acordado que, de maneira informal, seriam utilizadas algumas técnicas e/ou boas práticas que são comumente vistas e aplicadas em projetos que utilizam metodologia ágil.

[2] Os nomes citados no estudo de caso são fictícios, não foram utilizados os reais nomes das pessoas e/ou empresa envolvidos por questões de privacidade.

4.1 TÉCNICAS APLICADAS

Neste tópico serão apresentadas as técnicas elegidas para aplicação no estudo de caso, bem como os resultados obtidos.

4.1.1 Análise em pares.

Um dos pontos de gargalo e atrasos no processo estava na fase de análise e escrita dos casos de testes. De acordo com a metodologia da empresa, a documentação deve conter, com rigor, a maior quantidade de detalhes possíveis, mesmo com tempos curtos e constantes alterações nos documentos de requisitos (em certos casos a documentação era entregue depois das alterações realizadas na aplicação). Era comum ocorrer situações onde um cenário escrito por um analista era revisado e várias diferenças eram encontradas. Isso causava certo mal estar, uma vez que boa parte do trabalho tinha que ser realizada novamente. Como forma de reduzir este “retrabalho”, foi adotada a técnica de “programação em pares”, com alguns ajustes.

O trabalho de análise e escrita dos cenários de testes foi realizado em duplas, ou seja, o analista de testes lado a lado com o analista de requisitos. A cada alteração que a documentação sofria, o analista de teste em tempo real era informado das alterações e efetuava as modificações necessárias nos cenários correspondentes.

A aplicação desta técnica acarretou em uma melhora significativa na qualidade dos cenários e casos de testes desenvolvidos. Isso ficou perceptível na fase de execução dos testes, onde recursos de outras áreas foram alocados para auxiliar nos testes e não tiveram nenhuma dificuldade em executar os casos de testes (situação que antes não era comum). As revisões efetuadas posteriormente foram concluídas mais rapidamente, uma vez que poucas alterações foram necessárias.

4.1.2 *Sprint* na execução dos testes e divisão por funcionalidades.

O plano de execução de testes continha a divisão de atividades para cada testador. Era de conhecimento dos envolvidos o que e quando deveria ser entregue. Planilhas de controle e o plano de testes eram gravadas no servidor de CVS (*Concurrent Versions System*) e disponibilizadas para consulta a qualquer momento. Porém, nem sempre atividades correlacionadas eram agrupadas em uma mesma execução ou até mesmo para uma mesma pessoa. Essa distribuição acabava acarretando em morosidade na entrega dos testes, isso porque não era difícil ocorrerem situações onde dois testadores executavam testes muito similares. A entrega dos testes - efetuada por meio de relatório onde constam informações relativas à quantidade de testes executados, quantidade de defeitos encontrados, corrigidos/rejeitados, entre outros - era efetuada ao final da fase de testes como um todo, para em seguida o cliente efetuar o teste de aceite do software e evidências de testes.

Para melhorar o desempenho e entregas, optou-se por adotar as técnicas de *sprint* (utilizada no *SCRUM*) e divisão por funcionalidade (utilizada no FDD). O plano de execução passou a ser subdividido em tarefas por funcionalidades com uma atenção maior. Todos os testes relacionados a um mesmo tema (ou sua grande maioria) foram agrupados e atribuídos a uma única pessoa. Por exemplo, o testador responsável pela funcionalidade de pedidos de compra ficou responsável pelo processo como um todo, desde a geração do pedido até o recebimento do mesmo e geração de dados para o sistema de controle financeiro/fiscal. Para funcionalidades muito complexas (definidas e elencadas no levantamento de requisitos e destacadas como pontos de sucesso do projeto), as tarefas foram dividas em duas ou três pessoas, porém de maneira mais organizada, onde se tentava agrupar ao máximo as atividades, de forma a evitar repetições por pessoas diferentes.

As entregas também foram divididas. Semanalmente ou, em alguns casos, quinzenalmente, um determinado conjunto de testes já finalizados era entregue ao

cliente (por relatório, como citado anteriormente). As evidências de testes já finalizados eram aprovadas pelo cliente e, em seguida, o teste de aceite daquela funcionalidade já estava liberado para ser iniciado. Isso favoreceu a entrega final dos testes como um todo, uma vez que tarefas de execução e aprovação eram realizadas em paralelo. Funcionalidade entregue e aprovada era funcionalidade esquecida pelos testadores (pelo menos para aquele ciclo de provas). O cliente conseguia acompanhar e participar de forma ativa da evolução do projeto, uma vez que os relatórios de resultados passaram a ser entregues com maior frequência.

4.1.3 Reuniões em Pé (*Stand Up Meetings*) e Distribuição de Funções

A evolução do trabalho dos envolvidos nos testes era realizada por meio de planilhas de controle. Nestas planilhas os líderes e os gestores podiam acompanhar o andamento dos testes, podendo identificar gargalos e pontos que mereciam especial atenção. Porém a visibilidade dos problemas era limitada, uma vez que os problemas mais complexos demoravam a ser identificados. Muitas vezes o testador despendia do tempo de execução das atividades para procurar soluções para problemas. Os gestores nem sempre tinham visibilidade disso. Por exemplo, para que o teste fosse realizado com sucesso, o testador precisava que o cliente criasse registros no sistema. Para isso, o testador enviava solicitações para o cliente. Muitas vezes o cliente demorava tempo demais para responder e era obrigação do testador controlar e cobrar maior efetividade e rapidez nas respostas às solicitações. A FIGURA 6 representa parte deste controle:

	Tester	% OK	Observações
CN_005_RTV Synchronization			
CT_005_01_RTV Synchronization from RM10 to RM12. New RTV.	Tester 2	20%	Aguardando criação de dados pelo cliente
CT_005_02_RTV Synchronization from RM10 to RM12. Change RTV.	Tester 2	40%	Aguardando criação de dados pelo cliente
CT_005_04_RTV Synchronization from RD to RM10 - RM12. New RTV.	Tester 2	65%	Aguardando criação de dados pelo cliente
CT_005_05_RTV Synchronization from RD to RM10 - RM12. Change RTV.	Tester 2	50%	Aguardando criação de dados pelo cliente
CT_005_06_Store_RTV Synchronization from PS to RM10 - RM12. New RTV.	Tester 2	50%	Aguardando criação de dados pelo cliente
CT_005_07_RTV Synchronization from PS to RM10 - RM12. Change RTV.	Tester 2	50%	Aguardando criação de dados pelo cliente

FIGURA 6: PLANILHA DE CONTROLE DE EXECUÇÃO DE TESTES.
FONTE: Autor (2013).

Foi proposta e adotada uma melhor distribuição de atividades entre os integrantes do projeto. Assim como em metodologias ágeis, os lideres e gestores adotaram o papel de mediador, ou facilitador. Foi incluída em suas atividades a resolução, ou tentativa de resolução de problemas externos à atividade de testes (que antes podiam ser realizadas pelo próprio testador). Por exemplo, a demora na resposta de uma solicitação ao cliente, ou problemas na resolução de defeitos, passaram a ser atividades acompanhadas diariamente pelo coordenador de testes, que por sua vez recorria ao Gerente de Projetos para situações que estivessem fora de sua capacidade de resolução. Assim, qualquer atraso passou a ser identificado com antecedência e tratado de forma mais ágil, permitindo aos recursos de testes manterem o foco apenas na execução das atividades relativas aos testes, ao mesmo tempo em que permitiu aos gestores (coordenador, gerente de projetos e gestor de pessoas) maior proximidade e participação efetiva no projeto.

A FIGURA 7 exemplifica um relatório que representava solicitações pendentes, classificadas das mais atrasadas paras as mais recentes. Este relatório passou a ser enviado, duas vezes por dia, aos supervisores e gerentes. Estes, por sua vez, passaram a cobrar agilidade nas respostas.

ID	Categoría	Estado	Prioridad	Informador	Asignada a	Fecha de Envío	Actualizada	Resumen
1771	Satélites	asignada	normal	ea	qg	16/02/2012 11:09	20/02/2012	EAM - CT_005_01_JIT Validation - PMO integration
1774	RDM	asignada	normal	ea	qr	16/02/2012 16:16		EAM - CT_120_001_04_Warehouse Transfers - Envío y recepción de
1775	POS	asignada	urgente	am	js	16/02/2012 16:59		AMX - CT_115_002_14 Negative Inventory Adjustment(42) POS-RM
1777	RDM	asignada	normal	as	ag	16/02/2012 17:23		AES - CT_115_004_01_Manual Requisition RMS to RDM - PO RMS to
1796	Satélites	asignada	normal	am	ag	17/02/2012 15:54		EAM - CT_104_012_02_Validate XXFC-Módulo Adm. Redondeos
1802	POS	asignada	normal	as	js	17/02/2012 17:24		AES - CT_115_005_01_RTV POS to ORMS - Create one RTV in POS to
1805	POS	asignada	normal	pm	js	19/02/2012 19:26		PRM - CT_006_005_Validate_a_RTV (Create a RTV)
1807	RDM	asignada	normal	pm	ar	20/02/2012 07:06		PRM - CT_008_001_CEDIS_File_Conversion_With Programmed_sto
1810	RDM	asignada	normal	as	ar	20/02/2012 14:27		AES - CT_120_001_02_Centralized Transfers - Transfer Made by RM
1811	POS	asignada	normal	as	js	20/02/2012 10:47		AES - CT_115_004_02_Manual Requisition POS to RMS - Create a PO
1815	POS	asignada	urgente	am	js	20/02/2012 14:19		AMX - CT_115_002_11 Positive Inventory Adjustment(165) POS-RM
1816	POS	asignada	urgente	am	js	20/02/2012 14:30		AMX - CT_115_002_01 Positive Inventory Adjustment(150) POS-RM
1817	POS	asignada	alta	pm	js	20/02/2012 14:41		PRM - CT_001_01_Validates Inventory Adjustment from POS to OR
1818	Arquitetura	asignada	normal	hh	mg	20/02/2012 14:56		Investigación - Tablas utilizadas para el sincronismo de los ajustes
1820	POS	asignada	normal	as	js	20/02/2012 15:20		AES - CT_120_001_01_Store transfers - Create a Transfer Store to S
1822	POS	asignada	alta	em	js	20/02/2012 15:27		EAM - CT_120_001_01_Store transfers
1826	RDM	asignada	normal	pm	ar	20/02/2012 17:25		PRM - CT_004_001_Validate_IN_TRANSIT_QTY field - Store Requisi
1827	RDM	asignada	alta	em	ar	20/02/2012 17:56		EAM - CT_120_001_02_Centralized Transfers

FIGURA 7: RELATÓRIO DE SOLICITAÇÕES EM ATRASO.
FONTE: Autor (2013).

As reuniões em pé vieram muito a calhar. Diariamente, antes de iniciar as atividades, e em alguns casos antes do fim do expediente, reuniões rápidas e

objetivas eram realizadas. Cada integrante comentava o andamento de suas atividades, a previsão de entrega e, principalmente, os problemas passiveis de atrasos. Os líderes passaram assim a ter, além da planilha, as reuniões para acompanhamento. Muitos problemas puderam ser antecipados. Pôde ser percebida uma melhora significante na comunicação e até mesmo na integração entre as equipes de desenvolvimento (correção de defeitos) e de testes, assim como seus líderes e gestores. Problemas que antes tardavam a ser resolvidos passaram a ter uma resolução em tempo menor, uma vez que os problemas, assim como seus impactos, passaram a ser visualizados por todos os recursos envolvidos na fase de testes. Na FIGURA 8 é apresentado um exemplo de controle de ocorrências, apresentado nas reuniões. Por meio destas informações toda a equipe tinha visão dos problemas e mutuamente buscavam a resolução de forma mais ágil. Os SLA's[3] representam o tempo aceitável de resolução de cada problema. Os problemas eram agrupados por Categoria, que na planilha representava os sistemas legados ou áreas que eram responsáveis pela correção.

ID	Categoría	Estado	Fecha de Envío	SLA	Responsável
1815	POS	Aberto	20/02/2012 14:19	1 hora	am
1816	POS	Aberto	20/02/2012 14:30	1 hora	am
1817	POS	Fechado	20/02/2012 14:41	1 hora	pm
1818	Arquitetura	Novo	20/02/2012 14:56	4 horas	hh
1820	POS	Aberto	20/02/2012 15:20	4 horas	as
1822	POS	Fechado	20/02/2012 15:27	4 horas	em
1826	RDM	Novo	20/02/2012 17:25	6 horas	pm
1827	RDM	Aberto	20/02/2012 17:56	6 horas	em

FIGURA 8: CONTROLE DE OCORRÊNCIAS.
FONTE: Autor (2013).

[3] Service Level Agreement (SLA): documento que define padrões de um nível de serviços e a relação entre as partes: solicitador e o executor. (RUGGIERE, 2011).

4.1.4 Autonomia de Atividades

O planejamento das atividades desenvolvidas pela equipe de testes era definido no plano de testes. Porém, se alguma atividade não fosse passível de realização, era a líder da equipe quem normalmente reprogramava as atividades. Isso demandava tempo parado do recurso de teste enquanto era efetuado o replanejamento das atividades.

Foi sugerido que, nestas situações, o próprio testador poderia, dentro do seu cronograma e planejamento oficial, reprogramar as atividades de acordo com as necessidades e disponibilidade. Isto ajudou a diminuir o tempo ocioso dos recursos. Ao invés de aguardar a definição de suas próximas atividades, ele mesmo visualizava o que poderia ser antecipado e/ou adiado e, após aprovação, colocado em prática. A autonomia e confiança creditadas aos testadores ajudaram a manter o cronograma em dia ao mesmo tempo em que permitia ao líder de testes focar sua atenção à resolução de outros pontos de gargalo.

4.1.5 Ritmo Sustentável

Assim como em muitos projetos, seja por qual for o motivo, algumas das atividades eram realizadas fora do prazo. Era notório que o grande fluxo de atrasos estava em atividades externas à equipe de testes. Em quase 100% dos atrasos, o ponto de gargalo estava nas entregas de ambiente e / ou repostas às solicitações efetuadas ao cliente. Isso causava a necessidade de horas adicionais a fim de evitar atrasos nas entregas. Com a distribuição das atividades e autonomia de alterar as atividades de acordo com a necessidade, somadas a aplicação da metodologia do ritmo sustentável, foi possível reduzir, em média, 30% da necessidade de horas extras, mesmo com atrasos por parte do cliente.

4.1.6 Refatoração / Reutilização

Alterações em casos de testes ou cenários são pontos complicados de ser tratados. Muitas vezes é necessária a alteração do caso de testes antes da execução do mesmo. No caso da WSA não era diferente. Havia uma rotina definida e morosa para tais ações. Algumas vezes demandava-se um tempo razoável para correções dos passos, uma vez que era necessária a revisão dos analistas de testes, responsáveis pela escrita. Foi proposta a adoção das técnicas de reutilização e refatoração; assim, casos de testes similares, utilizados em outros projetos, puderam ser utilizados, bem como, em tempo de execução, alterações e aperfeiçoamentos puderam ser efetuados nos casos de testes. Como resultado, foram obtidas melhorias nas entregas, uma vez que a abrangência dos processos cobertos nos casos de testes foi tratada de maneira mais ampla.

4.1.7 Mural de Lembretes / *Post-Its*

Uma das mais conhecidas técnicas de metodologias ágeis, a utilização de lembretes referentes às atividades diárias e/ou semanais, foi adotada. Isso resultou em melhor controle e organização de tempos e execução de atividades planejadas. Os lembretes permitiram aos membros da equipe manter o foco nas atividades prioritárias, sem esquecer-se daquelas menos importantes. Igualmente, os lembretes ajudaram a equipe a ter uma visão macro das atividades de cada membro.

Um quadro de lembretes/atividades (FIGURA 9) foi fixado a uma das paredes, em local de fácil acesso e visível a toda a equipe. O quadro, dividido em colunas que representavam cada etapa dentro da fase de testes, serviu para fixação dos lembretes. Assim, a cada início ou término de atividade, um lembrete era colocado ou movido entre as colunas, conforme o avanço. A responsabilidade de manter os lembretes na coluna correta era do próprio executor da atividade.

Não foi utilizada convenção de cores, apenas padronizou-se o uso de Imãs, com as iniciais do nome do Executor como forma de identificar o dono da atividade.

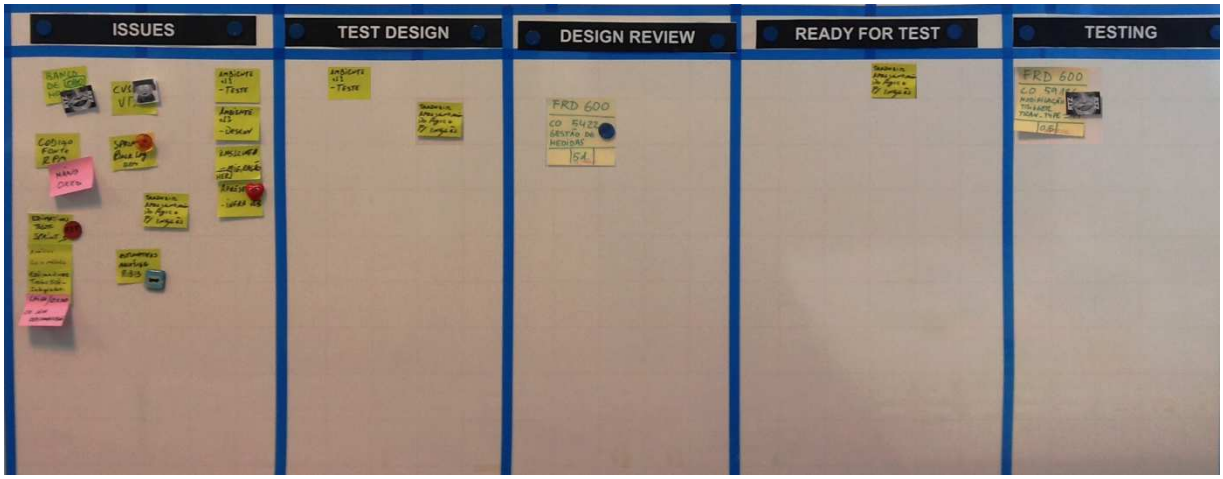

FIGURA 9: QUADRO DE LEMBRETES/ATIVIDADES.
FONTE: Autor (2013)

A aplicação da técnica facilitou a manutenção de atividades e ajudou na colaboração mútua. Por exemplo, quando um recurso X terminava suas atividades, pelo quadro de lembretes/atividades, era possível visualizar atividades pendentes, atribuída a outros recursos. Desse modo o recurso X assumia uma ou mais das atividades pendentes.

4.1.8 Testes exploratórios

Com a aplicação de várias técnicas ágeis no projeto de testes, muitas atividades foram executadas em menos tempo (aproximadamente 8% menos), se comparado com o planejado. Isto permitiu utilizar, mesmo que timidamente, o conceito de testes exploratórios. Funcionalidades que permitiam variações na execução, bem como funcionalidades não previstas no plano, foram inclusas em uma bateria de testes exploratórios. Foram encontradas algumas inconsistências que, sem os testes exploratórios, não seriam identificadas. Alguns exemplos:

- Teste de Pedido de Compras: Estava previsto teste de pedido normal. Foi aplicado teste exploratório, onde foi testado pedido com contratos (efetuados com fornecedores). Foi encontrado um erro não tratado para este tipo de pedido.
- Teste de Transferências: Após uma alteração que deveria apenas afetar novas transferências entre lojas de um determinado local, foram realizados testes exploratórios e foi identificado que a alteração foi aplicada também para transferências já existentes, fato que não estava no escopo da alteração.

Apenas pelo fato de evitar que o cliente encontre inconsistências que poderiam ser encontradas pela equipe de destes, já é um fato a se considerar. Identificá-las e corrigi-las dentro da fase de testes foi um feito bastante elogiado e comemorado internamente, ao final da fase de testes.

4.1.9 Reuniões de Revisão e Retrospectiva de S*print*

Mesmo na vida pessoal, os erros nos ensinam como agir em situações similares, que vierem a ocorrer no futuro. Profissionalmente não é diferente. Com isto em mente, foram adotadas as técnicas de reuniões de revisão e retrospectiva. A cada entrega efetuada ao cliente, a equipe se reunia e se concentrava em compartilhar as dificuldades, erros e êxitos obtidos. Ideias e planos de contingências foram debatidos e planejados para evitar que situações similares às anteriormente vividas fossem evitadas ou tratadas antes de se tornarem um risco maior ao projeto.

Do mesmo modo, ao fim da fase final dos testes, novo debate e ”*brainstorming*” foi realizado, com o intuito de aprendizado com acontecimentos vividos durante o projeto. Pontos positivos (como maior integração da equipe, envolvimento ativo dos gestores, maior participação do cliente) e negativos (pouca documentação formal gerada, horário de trabalho no mesmo fuso do cliente)

foram identificados, planos de contingência criados e, o mais importante, a maturidade, individual e da equipe, melhorada significativamente.

4.1.10 Análise de Viabilidade

Ao final da fase de testes abordada neste capítulo foi efetuada uma reunião, de cunho informal, para análise e avaliação de viabilidade de adoção das técnicas aplicadas em projetos futuros. Todos os envolvidos no projeto participaram e puderam dar a opinião, analisando os prós e contras de tudo o que foi proposto.

Após coleta das considerações de cada indivíduo, foi possível destacar como pontos positivos:

- Integração entre equipes (desenvolvimento, testes, gestores e cliente);
- Melhorias na comunicação;
- Envolvimento e participação ativa dos gestores na resolução de pontos críticos;
- Redução na carga de horas adicionais;
- Melhoras nos tempos de respostas de solicitações ao cliente e/ou correção de defeitos;
- Melhoria na organização e aplicação das atividades;
- Melhoria nos resultados;
- Satisfação do cliente (elogios formais escritos via *e-mail* e *feedback* positivo recebido pelo gerente de projeto);
- Aumento da maturidade da equipe e dos processos utilizados nos testes.

Do mesmo modo, os seguintes pontos negativos (ou passíveis de melhoria) foram identificados:

- Horário de trabalho diferente do horário de trabalho do cliente (questões de fuso);
- Dificuldade na comunicação com o cliente, como quedas de conexão em chamadas telefônicas ou aplicativo VOIP (*Voice Over Internet Protocol* ou Voz Sobre Protocolo de Internet);
- Prazos muitas vezes curtos para a atividade;
- Resistência, por partes de alguns (poucos) integrantes da equipe, no início das atividades.

De maneira geral, a aplicação das técnicas ágeis foi exitosa e os benefícios vistos por todos os integrantes. Apesar da resistência inicial, no decorrer das atividades a aceitação foi 100%.

Ficou decidido que, mesmo não sendo parte do processo formal de testes de software da companhia, as técnicas ágeis utilizadas poderiam ser adotadas nos demais projetos na medida em que fossem necessárias.

5 CONCLUSÃO

Muitos tópicos, mesmo que de maneira geral, foram abordados nesta pesquisa, e não é modéstia afirmar que temas antes desconhecidos para o leitor foram parcialmente esclarecidos.

Esta pesquisa fez uma abordagem conceitual sobre algumas das principais metodologias ágeis conhecidas, foram explicados os objetivos a que cada uma se aplica, assim como as técnicas e boas práticas utilizadas na concepção e aplicação em projetos de desenvolvimento de software.

Conceitos de testes de software foram explanados. Exemplos, tipos e técnicas de testes demonstrados.

Com base em todos os conceitos abordados, foi realizado um estudo de caso, onde foi possível avaliar a aplicação de técnicas ágeis em projetos específicos de testes, identificar ganhos, assim como levantar pontos positivos e negativos que justifiquem a viabilidade de adoção de tal estratégia.

Avaliando o estudo de caso e seus resultados, pode-se concluir que o objetivo desta pesquisa foi alcançado com êxito. Os resultados foram satisfatórios e a análise conclusiva é pela viabilidade de adoção de técnicas ágeis em projetos específicos de testes de software.

5.1 RECOMENDAÇÕES

Esta pesquisa foi elaborada visando não somente atingir o objetivo principal (já conhecido pelo leitor), mas também divulgar, e porque não apoiar, que a prática ágil seja adotada com mais frequência e aceitação por partes dos profissionais de testes de software. Dessa forma apresentam-se na sequência duas sugestões para trabalhos futuros que podem auxiliar neste objetivo adicional da presente pesquisa.

- Elaborar convenções comuns entre os integrantes da equipe de testes: o objetivo seria criar uma cultura ágil, mesmo que informal,

tornando assim uma prática global para projetos de testes, mesmo em ambientes onde os métodos e processos seguem padrões clássicos de engenharia de software.

- Abordagem Ágil focada em Qualidade e Custo: Avaliar, em termos de qualidade e custos, quão vantajosa seria a adoção formal de metodologias ágeis em projetos exclusivos de testes, deixando claro quais vantagens financeiras e qualitativas se pode alcançar quando um processo é ágil. Tal abordagem pode ser apresentada à alta gestão. Ou seja, números, documentos e resultados que provem aos gestores que a metodologia funciona bem para este nicho de negócio e que os ganhos não são vistos apenas na melhoria dos processos internos de testes, mas também nos resultados financeiros e de auditoria de qualidade da empresa.

REFERENCIAS

AGILE MANIFESTO. **Manifesto for Agile Software Development.** Disponível em: <http://www.agilemanifesto.org/>. Acesso em: 15 de dezembro de 2011.

BACH, James. **Exploratory Testing Explained.** 2003. Disponível em: <http://people.eecs.ku.edu/~saiedian/Teaching/Fa07/814/Resources/exploratory-testing.pdf>. Acesso em: 14 de novembro de 2012.

BARBOSA, Antonio; AZEVEDO, Bruno; PEREIRA, Bruno; CAMPOS, Pedro; SANTOS, Pedro. **Metodologia Ágil**: Feature Driven Development. Disponível em: <http://paginas.fe.up.pt/~aaguiar/es/artigos%20finais/es_final_22.pdf>. Acesso em: 14 de dezembro de 2011.

BARBOSA, Wladimir Henrique Mangelado. **A metodologia Extreme Programming: Um estudo de caso**. 2005. 25p. Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

CAETANO, Cristiano. **Testes Exploratórios de A a Z**. 2012. Disponível em: <http://www.linhadecodigo.com.br/artigo/1102/testes-exploratorios-de-a-a-z.aspx>. Acesso em: 14 de novembro de 2012.

CAMARA, Fabio. **Um cardápio de Metodologias Ágeis.** 2007. Disponível em: <http://imasters.com.br/artigo/7396/gerencia/um_cardapio_de_metodologias_ageis/>. Acesso em: 16 de dezembro de 2011.

CARVALHO, Afonso; SOUZA, Felipe de; BRITO, Rômulo de; BOAES, Wesley. **Extreme Programming**. 2011. 5p. Projeto Final – Organização da Informação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2011.

FAGUNDES, Priscila Basto; DETERS, Janice Inês; SANTOS, Sandro da Silva dos. **Comparação entre os Processos dos Métodos Ágeis:** XP, SCRUM, FDD E ASD em relação ao Desenvolvimento Iterativo Incremental. Florianópolis: E-Tech 2008.

HEPTAGON. FDD – **Feature Driven Development** – Descrição de Processos. 2012. Disponível em: <http://www.heptagon.com.br/fdd-estrutura>. Acesso em: 03 de março de 2012.

ISTQB. **Certified Tester Foundation Level Syllabus**. 2011. Disponível para download em <http://www.bstqb.org.br/?q=node/1181>. Acesso em: 13 de dezembro de 2011.

LIB – Lean Institute Brasil. **O Kanban e Lean Management**. 2010. Disponível em: <http://www.lean.org.br/leanmail/85/o-kanban-e-lean-management.aspx>. Acesso em: 16 janeiro de 2013.

LIBARDI, Paula L.O.; BARBOSA, Wladimir. **Métodos Ágeis.**
Limeira: Universidade Estadual de Campinas – UNICAMP, 2010.

MANER, Valter. **Rapid Application Development.** 1997. Disponível em: <http://csweb.cs.bgsu.edu/maner/domains/RAD.htm>. Acesso em: 13 de dezembro de 2011.

MARCONI, M.A.; LAKATOS,E.M.. **Metodologia do Trabalho Científico.** 6. ed. São Paulo: Atlas, 2001.

NETO, Arilo Claudio Dias. **Engenharia de Software Magazine – Introdução a Teste de Software.** Ano 1, 1. ed., 2007. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em: 13 de dezembro de 2012.

PAULK, Mark C. *A Scrum Adoption Survey*. **Software Quality Professional.** Milwaukee. v.15, n. 2, p. 27-34, março de 2013.

PINHEIRO , Cleber Wander Fernandes; SILVA, Fabrício Gomes da; CORREA, Francisco Juliano Silva; NETO, Moyses Cordeiro Vilaça; TAKAOKA, Vanessa de Lima; MIARALET, Lineu Fernando Stege. **Desenvolvendo Sistemas de Informação com Ferramentas RAD**. 2007. Universidade do Vale do Paraíba – Faculdade de Ciência da Computação, São José dos Campos.

PRESSMAN, Roger S. **Engenharia de Software.** São Paulo: Pearson, 2009.

ROCHA, Ana Regina Cavalcanti da; MALDONADO, José Carlos; WEBER, Kival Chaves. **Qualidade de software –** Teoria e prática. São Paulo: Prentice Hall, 2001.

RUGGIERE, Ruggero. **Como elaborar um SLA (The Service Level Agreement)**. 2011. Disponível em: < http://www.tiespecialistas.com.br/2011/01/como-elaborar-um-sla-the-service-level-agreement/#.UUnBqhc3uCQ>. Acesso em: 16 de dezembro de 2011.

SOARES, Michel dos Santos. **Metodologias Ágeis Extreme Programming e SCRUM para o desenvolvimento de Software.** Disponível em: <http://revistas.facecla.com.br/index.php/reinfo/article/view/146/38>. Acesso em: 15 de dezembro de 2011.

SOMMERVILLE, Ian. **Engenharia de Software.** 8. ed. São Paulo: Pearson, 2007.

TINKHAM, Andy; KANER, Cem. **Exploring Exploratory Testing**. 2003. Disponível em: <http://www.testingeducation.org/a/explore.pdf>. Acesso em: 14 de novembro de 2012.

VERSIONONE. **7th ANNUAL STATE of AGILE DEVELOPMENT SURVEY**. 2013. Disponível para download em <http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf>. Acesso em: 15 de abril de 2013.

DOCUMENTOS CONSULTADOS

BASTOS, Anderson; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. **Base de Conhecimento de Teste de Software.** Niterói: Traço e Fhoto, 2006.

CAMARA, Fabio. **Uma Metodologia Ágil – SCRUM.** Disponível em: <http://www.linhadecodigo.com.br/artigo/2084/uma-metodologia-agil-scrum.aspx>. Acesso em: 16 de dezembro de 2011.

CURSINO, Rodrigo. **Mini-Curso Teste Exploratório.** 2012. Disponível em: <http://www.gotest.biz/ebts2012/download/VI-EBTS-Mini-curso-TesteExploratorio.pdf>. Acesso em: 14 de novembro de 2012.

GOLDMAN, Alfredo; KON, Fabio. **Métodos Ágeis de Desenvolvimento de Software e a Programação eXtrema (XP)**. 2007. Disponível em: <http://ccsl.ime.usp.br/agilcoop/files/1-Introducao.pdf>. Acesso em: 16 de dezembro de 2011.

NADALETE, Lucas Gonçalves. **Teste Ágil, como implementar?** 2010. Disponível em: <http://dftestes.gershon.info/Capitulo7.html>. Acesso em: 15 de dezembro de 2011.

PAN, Jiantao. **Software Testing.** 1999. Disponível em: <http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/>. Acesso em: 13 de dezembro de 2012.

PEZZÈ, Mauro; YOUNG, Michal. **Teste e Análise de Software.** Processos, princípios e técnicas. Porto Alegre: Bookman, 2008.

QUALISTER. **Testes Ágeis.** 2011. Disponível em: <www.qualister.com.br>. Acesso em: 13 de dezembro de 2011.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de Software.** 2. ed. Rio de Janeiro: Alta Books, 2006.

SENE, Rafael Peria de. **Metodologia Ágil -** O SCRUM de forma simples. 2011. Disponível em <http://www.profissionaisti.com.br/2011/09/metodologia-agil-o-scrum-de-forma-simples/>. Acesso em 13 de dezembro de 2011.

SOTILLE, Mauro. **Gerenciamento de Projetos na Engenharia de Software.** 2004. Disponível em: <http://www.pmtech.com.br/artigos/Gerenciamento_Projetos_Software.pdf>. Acesso em: 16 de dezembro de 2011.

VIEIRA, L. A.. **Projeto de pesquisa e monografia o que é? Como se faz? Normas da ABNT**. 2. ed. Curitiba: Ed. Do autor, 2002.

UNIVERSIDADE FEDERAL DO PARANÁ, Sistema de Bibliotecas. **Teses, dissertações, monografias e trabalhos acadêmicos**. Curitiba: Editora da UFPR,
2007.